ENERGY MINIMIZATION FOR CLASSIFICATION, PATTERN RECOGNITION, SENSOR FUSION, DATA COMPRESSION, NETWORK RECONSTRUCTION AND SIGNAL PROCESSING

CROSS REFERENCE TO RELATED APPLICATIONS

This application claims priority of U.S. provisional application serial number 60/071,592, filed December 29, 1997.

COPYRIGHT NOTICE

APPENDIX

An appendix of computer program source code is included and comprises 22 sheets.

The Appendix is hereby expressly incorporated herein by reference, and contains material which is subject to copyright protection as set forth above.

BACKGROUND OF THE INVENTION

The present invention relates to recognition, analysis, and classification of patterns in data from real world sources, events and processes. Patterns exist throughout the real world. Patterns also exist in the data used to represent or convey or store information about real world objects or events or processes. As information systems process more real world data, there are mounting requirements to build more sophisticated, capable and reliable pattern recognition systems.

Existing pattern recognition systems include statistical, syntactic and neural systems. Each of these systems has certain strengths which lends it to specific applications. Each of these systems has problems which limit its effectiveness.

Existing pattern recognition systems include statistical, syntactic and neural systems. Each of these systems has certain strengths which lends it to specific applications. Each of these systems has problems which limit its effectiveness.

Some real world patterns are purely statistical in nature. Statistical and probabilistic pattern recognition works by expecting data to exhibit statistical patterns. Pattern recognition by this method alone is limited. Statistical pattern recognizers cannot see beyond the expected statistical pattern. Only the expected statistical pattern can be detected.

Syntactic pattern recognizers function by expecting data to exhibit structure. While syntactic pattern recognizers are an improvement over statistical pattern recognizers, perception is still narrow and the system cannot perceive beyond the expected structures. While some real world patterns are structural in nature, the extraction of structure is unreliable.

Pattern recognition systems that rely upon neural pattern recognizers are an improvement over statistical and syntactic recognizers. Neural recognizers operate by storing training patterns as synaptic weights. Later stimulation retrieves these patterns and classifies the data. However, the fixed structure of neural pattern recognizers limits their scope of recognition. While a neural system can learn on its own, it can only find the patterns that its fixed structure allows it to see. The difficulties with this fixed structure are illustrated by the well-known problem that the number of hidden layers in a neural network strongly affects its ability to learn and generalize. Additionally, neural pattern recognition results are often not reproducible. Neural nets are also sensitive to training order, often require redundant data for training, can be slow learners and sometimes never learn. Most importantly, as with statistical and syntactic pattern recognition systems, neural pattern recognition systems are incapable of discovering truly new knowledge.

Accordingly, there is a need for an improved method and apparatus for pattern recognition, analysis, and classification which is not encumbered by preconceptions about data or models.

BRIEF SUMMARY OF THE INVENTION

By way of illustration only, an analyzer/classifier process for data comprises using energy-minimization with one or more input matrices. The data to be analyzed/classified is processed by an energy minimization technique such as individual differences multidimensional scaling (IDMDS) to produce at least a rate of change of stress/energy. Using the rate of change of stress/energy and possibly other IDMDS output, the data are analyzed or classified through patterns recognized within the data. The foregoing discussion of one embodiment has been presented only by way of introduction. Nothing in this section should be taken as a limitation on the following claims, which define the scope of the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a diagram illustrating components of an analyzer according to the first embodiment of the invention; and

FIG. 2 through FIG. 10 relate to examples illustrating use of an embodiment of the invention for data classification, pattern recognition, and signal processing.

DETAILED DESCRIPTION THE PRESENTLY PREFERRED EMBODIMENTS

The method and apparatus in accordance with the present invention provide an analysis tool with many applications. This tool can be used for data classification, pattern recognition, signal processing, sensor fusion, data compression, network reconstruction, and many other purposes. The invention relates to a general method for data analysis based on energy minimization and least energy deformations. The invention uses energy minimization principles to analyze one to many data sets. As used herein, energy is a convenient descriptor for concepts which are handled similarly mathematically. Generally, the physical concept of energy is not intended by use of this term but the more general mathematical concept. Within multiple data sets, individual data sets are

characterized by their deformation under least energy merging. This is a contextual characterization which allows the invention to exhibit integrated unsupervised learning and generalization. A number of methods for producing energy minimization and least energy merging and extraction of deformation information have been identified; these include, the finite element method (FEM), simulated annealing, and individual differences multidimensional scaling (IDMDS). The presently preferred embodiment of the invention utilizes individual differences multidimensional scaling (IDMDS).

Multidimensional scaling (MDS) is a class of automated, numerical techniques for converting proximity data into geometric data. IDMDS is a generalization of MDS, which converts multiple sources of proximity data into a common geometric configuration space, called the common space, and an associated vector space called the source space. Elements of the source space encode deformations of the common space specific to each source of proximity data. MDS and IDMDS were developed for psychometric research, but are now standard tools in many statistical software packages. MDS and IDMDS are often described as data visualization techniques. This description emphasizes only one aspect of these algorithms.

Broadly, the goal of MDS and IDMDS is to represent proximity data in a low dimensional metric space. This has been expressed mathematically by others (see, for example, de Leeuw, J. and Heiser, W., "Theory of multidimensional scaling," in P. R. Krishnaiah and L. N. Kanal, eds., *Handbook of Statistics*, Vol. 2. North-Holland, New York, 1982) as follows. Let S be a nonempty finite set, p a real valued function on $S \times S$,

$$p : S \times S \rightarrow \boldsymbol{R}.$$

p is a measure of proximity between objects in S. Then the goal of MDS is to construct a mapping $\tilde{f}$ from S into a metric space (X, d),

$$\tilde{f} : S \rightarrow X,$$

such that $p(i,j) = p_{ij} \approx d(\tilde{f}(i), \tilde{f}(j))$, that is, such that the proximity of object i to object j in S is approximated by the distance in X between $\tilde{f}(i)$ and $\tilde{f}(j)$. X is usually assumed to be n dimensional Euclidean space $\boldsymbol{R}^n$, with n sufficiently small.

IDMDS generalizes MDS by allowing multiple sources. For $k = 1,\ldots,m$ let S_k be a finite set with proximity measure p_k, then IDMDS constructs maps

$$\tilde{f}_k : S_k \to X$$

such that $p_k(i,j) = p_{ijk} \approx d(\tilde{f}_k(i), \tilde{f}_k(j))$, for $k = 1,\ldots,m$.

Intuitively, IDMDS is a method for representing many points of view. The different proximities p_k can be viewed as giving the proximity perceptions of different judges. IDMDS accommodates these different points of view by finding different maps $\tilde{f}_k$ for each judge. These individual maps, or their image configurations, are deformations of a common configuration space whose interpoint distances represent the common or merged point of view.

MDS and IDMDS can equivalently be described in terms of transformation functions. Let $P = (p_{ij})$ be the matrix defined by the proximity p on $S \times S$. Then MDS defines a transformation function

$$f : p_{ij} \mapsto d_{ij}(X),$$

where $d_{ij}(X) = d(\tilde{f}(i), \tilde{f}(j))$, with $\tilde{f}$ the mapping from $S \to X$ induced by the transformation function f. Here, by abuse of notation, $X = \tilde{f}(S)$, also denotes the image of S under $\tilde{f}$. The transformation function f should be optimal in the sense that the distances $f(p_{ij})$ give the best approximation to the proximities p_{ij}. This optimization criterion is described in more detail below. IDMDS is similarly re-

expressed; the single transformation f is replaced by m transformations f_k. Note, these f_k need not be distinct. In the following, the image of S_k under f_k will be written X_k.

MDS and IDMDS can be further broken down into so-called metric and nonmetric versions. In metric MDS or IDMDS, the transformations f (f_k) are parametric functions of the proximities $p_{ij}(p_{ijk})$. Nonmetric MDS or IDMDS generalizes the metric approach by allowing arbitrary admissible transformations f (f_k), where admissible means the association between proximities and transformed proximities (also called disparities in this context) is weakly monotone:

$$p_{ij} < p_{kl} \text{ implies } f(p_{ij}) \leq f(p_{kl}).$$

Beyond the metric-nonmetric distinction, algorithms for MDS and IDMDS are distinguished by their optimization criteria and numerical optimization routines. One particularly elegant and publicly available IDMDS algorithm is PROXSCAL See Commandeur, J. and Heiser, W., "Mathematical derivations in the proximity scaling (PROXSCAL) of symmetric data matrices," *Tech. report no. RR-93-03*, Department of Data Theory, Leiden University, Leiden, The Netherlands. PROXSCAL is a least squares, constrained majorization algorithm for IDMDS.We now summarize this algorithm, following closely the above reference.

PROXSCAL is a least squares approach to IDMDS which minimizes the objective function

$$\sigma(f_1,\ldots,f_m,X_1,\ldots,X_m) = \sum_{k=1}^{m}\sum_{i<j}^{n} w_{ijk}(f_k(p_{ijk}) - d_{ij}(X_k))^2 .$$

σ is called the stress and measures the goodness-of-fit of the configuration distances $d_{ij}(X_k)$ to the transformed proximities $f_k(p_{ijk})$. This is the most general form for the objective function. MDS can be interpreted as an energy minimization process and stress can be interpreted as an energy functional. The w_{ijk} are proximity weights. For simplicity, it is assumed in what follows that $w_{ijk} = 1$ for all i, j, k.

The PROXSCAL majorization algorithm for MDS with transformations is summarized as follows.

1. Choose a (possibly random) initial configuration X^0.
2. Find optimal transformations $f(p_{ij})$ for fixed distances $d_{ij}(X^0)$.
3. Compute the initial stress

$$\sigma(f, X^0) = \sum_{i<j}^{n} (f(p_{ij}) - d_{ij}(X^0))^2.$$

4. Compute the Guttman transform $\overline{X}$ of X^0 with the transformed proximities $f(p_{ij})$. This is the majorization step.
5. Replace X^0 with $\overline{X}$ and find optimal transformations $f(p_{ij})$ for fixed distances $d_{ij}(X^0)$.
6. Compute $\sigma(f, X^0)$.
7. Go to step 4 if the difference between the current and previous stress is not less than ε, some previously defined number. Stop otherwise.

For multiple sources of proximity data, restrictions are imposed on the configurations X_k associated to each source of proximity data in the form of the constraint equation $X_k = ZW_k$.

This equation defines a common configuration space Z and diagonal weight matrices W_k. Z represents a merged or common version of the input sources, while the W_k define the deformation of the common space required to produce the

individual configurations X_k. The vectors defined by diag(W_k), the diagonal entries of the weight matrices W_k, form the source space W associated to the common space Z.

The PROXSCAL constrained majorization algorithm for IDMDS with transformations is summarized as follows. To simplify the discussion, so-called unconditional IDMDS is described. This means the m transformation functions are the same: $f_1 = f_2 = \cdots = f_m$.

1. Choose constrained initial configurations X_k^0.
2. Find optimal transformations $f(p_{ijk})$ for fixed distances $d_{ij}(X_k^0)$.
3. Compute the initial stress

$$\sigma(f, X_1^0, \ldots, X_m^0) = \sum_{k=1}^{m} \sum_{i<j}^{n} (f(p_{ijk}) - d_{ij}(X_k^0))^2.$$

4. Compute unconstrained updates $\bar{X}_k$ of X_k^0 using the Guttman transform with transformed proximities $f(p_{ijk})$. This is the unconstrained majorization step.
5. Solve the metric projection problem by finding X_k^+ minimizing

$$h(X_1, \ldots, X_m) = \sum_{k=1}^{m} \mathrm{tr}\,(X_k - \bar{X}_k)'(X_k - \bar{X}_k)$$

subject to the constraints $X_k = ZW_k$. This step constrains the updated configurations from step 4.
6. Replace X_k^0 with X_k^+ and find optimal transformations $f(p_{ijk})$ for fixed distances $d_{ij}(X_k^0)$.
7. Compute $\sigma(f, X_1^0, \ldots, X_m^0)$.

8. Go to step 4 if the difference between the current and previous stress is not less than ε, some previously defined number. Stop otherwise.

Here, tr(A) and A' denote, respectively, the trace and transpose of matrix A.

It should be pointed out that other IDMDS routines do not contain an explicit constraint condition. For example, ALSCAL (see Takane, Y., Young, F, and de Leeuw, J., "Nonmetric individual differences multidimensional scaling: an alternating least squares method with optimal scaling features," *Psychometrika*, Vol. 42, 1977) minimizes a different energy expression (sstress) over transformations, configurations, and weighted Euclidean metrics. ALSCAL also produces common and source spaces, but these spaces are computed through alternating least squares without explicit use of constraints. Either form of IDMDS can be used in the present invention.

MDS and IDMDS have proven useful for many kinds of analyses. However, it is believed that prior utilizations of these techniques have not extended the use of these techniques to further possible uses for which MDS and IDMDS have particular utility and provide exceptional results. Accordingly, one benefit of the present invention is to incorporate MDS or IDMDS as part of a platform in which aspects of these techniques are extended. A further benefit is to provide an analysis technique, part of which uses IDMDS, that has utility as an analytic engine applicable to problems in classification, pattern recognition, signal processing, sensor fusion, and data compression, as well as many other kinds of data analytic applications.

Referring now to FIG. 1, it illustrates an operational block diagram of a data analysis/classifier tool 100. The least energy deformation analyzer/classifier is a three-step process. Step 110 is a front end for data transformation. Step 120 is a process step implementing energy minimization and deformation computations—in the presently preferred embodiment, this process step is implemented through the IDMDS algorithm. Step 130 is a back end which

interprets or decodes the output of the process step 120. These three steps are illustrated in FIG. 1.

It is to be understood that the steps forming the tool 100 may be implemented in a computer usable medium or in a computer system as computer executable software code. In such an embodiment, step 110 may be configured as a code, step 120 may be configured as second code and step 120 may be configured as third code, with each code comprising a plurality of machine readable steps or operations for performing the specified operations. While step 110, step 120 and step 130 have been shown as three separate elements, their functionality can be combined and/or distributed. It is to be further understood that "medium" is intended to broadly include any suitable medium, including analog or digital, hardware or software, now in use or developed in the future.

Step 110 of the tool 100 is the transformation of the data into matrix form. The only constraint on this transformation for the illustrated embodiment is that the resulting matrices be square. The type of transformation used depends on the data to be processed and the goal of the analysis. In particular, it is not required that the matrices be proximity matrices in the traditional sense associated with IDMDS. For example, time series and other sequential data may be transformed into source matrices through straight substitution into entries of symmetric matrices of sufficient dimensionality (this transformation will be discussed in more detail in an example below). Time series or other signal processing data may also be Fourier or otherwise analyzed and then transformed to matrix form.

Step 120 of the tool 100 implements energy minimization and extraction of deformation information through IDMDS. In the IDMDS embodiment of the tool 100, the stress function σ defines an energy functional over configurations and transformations. The configurations are further restricted to those which satisfy the constraint equations $X_k = ZW_k$. For each configuration X_k, the weight vectors diag(W_k) are the contextual signature, with respect to the common space, of the k-th input source. Interpretation of σ as an energy functional is

fundamental; it greatly expands the applicability of MDS as an energy minimization engine for data classification and analysis.

Step 130 consists of both visual and analytic methods for decoding and interpreting the source space W from step 120. Unlike traditional applications of IDMDS, tool 100 often produces high dimensional output. Among other things, this makes visual interpretation and decoding of the source space problematic. Possible analytic methods for understanding the high dimensional spaces include, but are not limited to, linear programming techniques for hyperplane and decision surface estimation, cluster analysis techniques, and generalized gravitational model computations. A source space dye-dropping or tracer technique has been developed for both source space visualization and analytic postprocessing. Step 130 may also consist in recording stress/energy, or the rate of change of stress/energy, over multiple dimensions. The graph of energy (rate or change or stress/energy) against dimension can be used to determine network and dynamical system dimensionality. The graph of stress/energy against dimensionality is traditionally called a scree plot. The use and purpose of the scree plot is greatly extended in the present embodiment of the tool 100.

Let $S = \{S_k\}$ be a collection of data sets or sources S_k for $k = 1,\ldots,m$. Step 110 of the tool 100 converts each $S_k \in S$ to matrix form $M(S_k)$ where $M(S_k)$ is a p dimensional real hollow symmetric matrix. Hollow means the diagonal entries of $M(S_k)$ are zero. As indicated above, $M(S_k)$ need not be symmetric or hollow, but for simplicity of exposition these additional restrictions are adopted. Note also that the matrix dimensionality p is a function of the data S and the goal of the analysis. Since $M(S_k)$ is hollow symmetric, it can be interpreted and processed in IDMDS as a proximity (dissimilarity) matrix. Step 110 can be represented by the map

$$M : S \to H^p(\boldsymbol{R}),$$
$$S_k \mapsto M(S_k)$$

where $H^p(\boldsymbol{R})$ is the set of p dimensional hollow real symmetric matrices. The precise rule for computing M depends on the type of data in S, and the purpose of the analysis. For example, if S contains time series data, then M might entail the straightforward entry-wise encoding mentioned above. If S consists of optical character recognition data, or some other kind of geometric data, then $M(S_k)$ may be a standard distance matrix whose ij-th entry is the Euclidean distance between "on" pixels i and j. M can also be combined with other transformations to form the composite, $(M \circ F)(S_k)$, where F, for example, is a fast Fourier transform (FFT) on signal data S_k. To make this more concrete, in the examples below M will be explicitly calculated in a number of different ways. It should also be pointed out that for certain data collections S it is possible to analyze the conjugate or transpose S^t of S. For instance, in data mining applications, it is useful to transpose records (clients) and fields (client attributes) thus allowing analysis of attributes as well as clients. The mapping M is simply applied to the transposed data.

Step 120 of the presently preferred embodiment of the tool 100 is the application of IDMDS to the set of input matrices $M(S) = \{M(S_k)\}$. Each $M(S_k) \in M(S)$ is an input source for IDMDS. As described above, the IDMDS output is a common space $Z \subset \boldsymbol{R}^n$ and a source space W. The dimensionality n of these spaces depends on the input data S and the goal of the analysis. For signal data, it is often useful to set $n = p - 1$ or even $n = |S_k|$ where $|S_k|$ denotes the cardinality of S_k. For data compression, low dimensional output spaces are essential. In the case of network reconstruction, system dimensionality is discovered by the invention itself.

IDMDS can be thought of as a constrained energy minimization process. As discussed above,the stress σ is an energy functional defined over transformations and configurations in $\boldsymbol{R}^n$; the constraints are defined by the constraint equation $X_k = ZW_k$. IDMDS attempts to find the lowest stress or energy configurations X_k which also satisfy the constraint equation. (MDS is the

special case when each $W_k = I$,the identity matrix.) Configurations X_k most similar to the source matrices $M(S_k)$ have the lowest energy. At the same time, each X_k is required to match the common space Z up to deformation defined by the weight matrices W_k. The common space serves as a characteristic, or reference object. Differences between individual configurations are expressed in terms of this characteristic object with these differences encoded in the weight matrices W_k. The deformation information contained in the weight matrices, or, equivalently, in the weight vectors defined by their diagonal entries, becomes the signature of the configurations X_k and hence the sources S_k (through $M(S_k)$). The source space may be thought of as a signature classification space.

The weight space signatures are contextual; they are defined with respect to the reference object Z. The contextual nature of the source deformation signature is fundamental. As the polygon classification example below will show, Z-contextuality of the signature allows the tool 100 to display integrated unsupervised machine learning and generalization. The analyzer/classifier learns seamlessly and invisibly. Z-contextuality also allows the tool 100 to operate without a priori data models. The analyzer/classifier constructs its own model of the data, the common space Z.

Step 130, the back end of the tool 100, decodes and interprets the source or classification space output W from IDMDS. Since this output can be high dimensional, visualization techniques must be supplemented by analytic methods of interpretation. A dye-dropping or tracer technique has been developed for both visual and analytic postprocessing. This entails differential marking or coloring of source space output. The specification of the dye-dropping is contingent upon the data and overall analysis goals. For example, dye-dropping may be two-color or binary allowing separating hyperplanes to be visually or analytically determined. For an analytic approach to separating hyperplanes using binary dye-dropping see Bosch, R. and Smith, J, "Separating hyperplanes and the authorship of the disputed federalist papers," *American Mathematical Monthly*, Vol. 105, 1998.

Discrete dye-dropping allows the definition of generalized gravitational clustering measures of the form

$$g_p(A,x) = \frac{\sum_{y \neq x} \chi_A(x) \exp(p \cdot d(x,y))}{\sum_{y \neq x} \exp(p \cdot d(x,y))}.$$

Here, A denotes a subset of W (indicated by dye-dropping), $\chi_A(x)$, is the characteristic function on A, $d(\cdot,\cdot)$ is a distance function, and $p \in R$. Such measures may be useful for estimating missing values in data bases. Dye-dropping can be defined continuously, as well, producing a kind of height function on W. This allows the definition of decision surfaces or volumetric discriminators. The source space W is also analyzable using standard cluster analytic techniques. The precise clustering metric depends on the specifications and conditions of the IDMDS analysis in question.

Finally, as mentioned earlier, the stress/energy and rate of change of stress/energy can be used as postprocessing tools. Minima or kinks in a plot of energy, or the rate of change of energy, over dimension can be used to determine the dimensionality of complex networks and general dynamical systems for which only partial output information is available. In fact, this technique allows dimensionality to be inferred often from only a single data stream of time series of observed data.

A number of examples are presented below to illustrate the method and apparatus in accordance with the present invention. These examples are illustrative only and in no way limit the scope of the method or apparatus.

Example A: Classification of regular polygons

The goal of this experiment was to classify a set of regular polygons. The collection $S = \{S_1, \ldots, S_{16}\}$ with data sets $S_1 - S_4$, equilateral triangles; $S_5 - S_8$, squares; $S_9 - S_{12}$, pentagons; and $S_{13} - S_{16}$; hexagons. Within each subset of distinct polygons, the size of the figures is increasing with the subscript. The perimeter of each polygon S_k was divided into 60 equal segments with the

segment endpoints ordered clockwise from a fixed initial endpoint. A turtle application was then applied to each polygon to compute the Euclidean distance from each segment endpoint to every other segment endpoint (initial endpoint included). Let $x^i_{S_k}$ denote the i-th endpoint of polygon S_k, then the mapping M is defined by

$$M : S \to H^{60}(\boldsymbol{R}),$$

$$S_k \mapsto [d_{S_k^1} \mid d_{S_k^2} \mid \cdots \mid d_{S_k^{60}}]$$

where the columns

$$d_{S_k^i} = (d(x^i_{S_k}, x^1_{S_k}), d(x^i_{S_k}, x^2_{S_k}), \ldots, d(x^i_{S_k}, x^{60}_{S_k}))^t .$$

The individual column vectors $d_{S_k^i}$ have intrinsic interest. When plotted as functions of arc length they represent a geometric signal which contains both frequency and spatial information.

The 16, 60×60 distance matrices were input into a publicly distributed version of PROXSCAL. PROXSCAL was run with the following technical specifications: sources- 16, objects- 60, dimension- 4, model- weighted, initial configuration- Torgerson, conditionality- unconditional, transformations- numerical, rate of convergence- 0.0, number of iterations- 500, and minimum stress- 0.0.

FIG. 2 and FIG. 3 show the four dimensional common and source space output. The common space configuration appears to be a multifaceted representation of the original polygons. It forms a simple closed path in four dimensions which, when viewed from different angles, or, what is essentially the same thing, when deformed by the weight matrices, produces a best, in the sense of minimal energy, representation of each of the two dimensional polygonal

figures. The most successful such representation appears to be that of the triangle projected onto the plane determined by dimensions 2 and 4.

In the source space, the different types of polygons are arranged, and hence, classified, along different radii. Magnitudes within each such radial classification indicate polygon size or scale with the smaller polygons located nearer the origin.

The contextual nature of the polygon classification is embodied in the common space configuration. Intuitively, this configuration looks like a single, carefully bent wire loop. When viewed from different angles, as encoded by the source space vectors, this loop of wire looks variously like a triangle, a square, a pentagon, or a hexagon.

Example B: Classification of non-regular polygons

The polygons in Example A were regular. In this example, irregular polygons $S = \{S_1,\ldots,S_6\}$ are considered, where $S_1 - S_3$ are triangles and $S_4 - S_6$ rectangles. The perimeter of each figure S_k was divided into 30 equal segments with the preprocessing transformation M computed as in Example A. This produced 6, 30×30 source matrices which were input into PROXSCAL with technical specifications the same as those above except for the number of sources, 6, and objects, 30.

FIG. 4 and FIG. 5 show the three dimensional common and source space outputs. The common space configuration, again, has a "holographic" or faceted quality; when illuminated from different angles, it represents each of the polygonal figures. As before, this change of viewpoint is encoded in the source space weight vectors. While the weight vectors encoding triangles and rectangles are no longer radially arranged, they can clearly be separated by a hyperplane and are thus accurately classified by the analysis tool as presently embodied.

It is notable that two dimensional IDMDS outputs were not sufficient to classify these polygons in the sense that source space separating hyperplanes did not exist in two dimensions.

Example C: Time series data

This example relates to signal processing and demonstrates the analysis tool's invariance with respect to phase and frequency modification of time series data. It also demonstrates an entry-wise approach to computing the preprocessing transformation *M*.

The set $S = \{S_1, \ldots, S_{12}\}$ consisted of sine, square, and sawtooth waveforms. Four versions of each waveform were included, each modified for frequency and phase content. Indices 1-4 indicate sine, 5-8 square, and 9-12 sawtooth frequency and phase modified waveforms. All signals had unit amplitude and were sampled at 32 equal intervals x, for $0 \le x \le 2\pi$.

Each time series S_k was mapped into a symmetric matrix as follows. First, an "empty" nine dimensional, lower triangular matrix $T_k = (t_{ij}^k) = T(S_k)$ was created. "Empty" meant that T_k had no entries below the diagonal and zeros everywhere else. Nine dimensions were chosen since nine is the smallest positive integer m satisfying the inequality $m(m-1)/2 \ge 32$ and $m(m-1)/2$ is the number of entries below the diagonal in an m dimensional matrix. The empty entries in T_k were then filled in, from upper left to lower right, column by column, by reading in the time series data from S_k. Explicitly: $s_1^k = t_{21}^k$, the first sample in S_k was written in the second row, first column of T_k; $s_2^k = t_{31}^k$, the second sample in S_k was written in the third row, first column of T_k, and so on. Since there were only 32 signal samples for 36 empty slots in T_k, the four remaining entries were designated missing by writing –2 in these positions (These entries are then ignored when calculating the stress). Finally, a hollow symmetric matrix was defined by setting

$$M(S_k) = T_k + T_k'.$$

This preprocessing produced 12, 9×9 source matrices which were input to PROXSCAL with the following technical specifications: sources- 12, objects- 9,

dimension- 8, model- weighted, initial configuration- Torgerson, conditionality- unconditional, transformations- ordinal, approach to ties- secondary, rate of convergence- 0.0, number of iterations- 500, and minimum stress- 0.0. Note that the data, while metric or numeric, was transformed as if it were ordinal or nonmetric. The use of nonmetric IDMDS has been greatly extended in the present embodiment of the tool 100.

FIG. 6 shows the eight dimensional source space output for the time series data. The projection in dimensions seven and eight, as detailed in FIG. 7, shows the input signals are separated by hyperplanes into sine, square, and sawtooth waveform classes independent of the frequency or phase content of the signals.

Example D: Sequences, Fibonacci, etc.

The data set $S = \{S_1,\ldots,S_9\}$ in this example consisted of nine sequences with ten elements each; they are shown in Table 1, FIG. 8. Sequences 1-3 are constant, arithmetic, and Fibonacci sequences respectively. Sequences 4-6 are these same sequences with some error or noise introduced. Sequences 7-9 are the same as 1-3, but the negative 1's indicate that these elements are missing or unknown.

The nine source matrices $M(S_k) = (m_{ij}^k)$ were defined by

$$m_{ij}^k = \left| s_i^k - s_j^k \right|,$$

the absolute value of the difference of the i-th and j-th elements in sequence S_k. The resulting 10×10 source matrices where input to PROXSCAL configured as follows: sources- 9, objects- 10, dimension- 8, model- weighted, initial configuration- simplex, conditionality- unconditional, transformations- numerical, rate of convergence- 0.0, number of iterations- 500, and minimum stress- 0.0.

FIG. 9 shows dimensions 5 and 6 of the eight dimensional source space output. The sequences are clustered, hence classified, according to whether they are constant, arithmetic, or Fibonacci based. Note that in this projection, the constant sequence and the constant sequence with missing element coincide,

therefore only two versions of the constant sequence are visible. This result demonstrates that the tool 100 of the presently preferred embodiment can function on noisy or error containing, partially known, sequential data sets.

Example E: Missing value estimation for bridges

This example extends the previous result to demonstrate the applicability of the analysis tool to missing value estimation on noisy, real-world data. The data set consisted of nine categories of bridge data from the National Bridge Inventory (NBI) of the Federal Highway Administration. One of these categories, bridge material (steel or concrete), was removed from the database. The goal was to repopulate this missing category using the technique of the presently preferred embodiment to estimate the missing values.

One hundred bridges were arbitrarily chosen from the NBI. Each bridge defined an eight dimensional vector of data with components the NBI categories. These vectors were preprocessed as in Example D, creating one hundred 8×8 source matrices. The matrices were submitted to PROXSCAL with specifications: sources- 100, objects- 8, dimension- 7, model- weighted, initial configuration- simplex, conditionality- unconditional, transformations- numerical, rate of convergence- 0.0, number of iterations- 500, and minimum stress- 0.00001.

The seven dimensional source space output was partially labeled by bridge material—an application of dye-dropping—and analyzed using the following function

$$g_p(A_i, x) = \frac{\sum_{y \neq x} \chi_{A_i}(x) \cdot d(x,y)^{-p}}{\sum_{y \neq x} d(x,y)^{-p}}$$

where p is an empirically determined negative number, $d(x,y)$ is Euclidean distance on the source space, and χ_{A_i} is the characteristic function on material set A_i, $i = 1,2$, where A_1 is steel, A_2 concrete. (For the bridge data, no two bridges had the same source space coordinates, hence g_p was well-defined.) A bridge

was determined to be steel (concrete) if $g_p(A_1,x) > g_p(A_2,x)$ ($g_p(A_1,x) < g_p(A_2,x)$). The result was indeterminate in case of equality.

The tool 100 illustrated in FIG. 1 estimated bridge construction material with 90 percent accuracy.

Example F: Network dimensionality for a 4-node network

This example demonstrates the use of stress/energy minima to determine network dimensionality from partial network output data. Dimensionality, in this example, means the number of nodes in a network.

A four-node network was constructed as follows: generator nodes 1 to 3 were defined by the sine functions, $\sin(2x)$, $\sin(2x+\frac{\pi}{2})$, and $\sin(2x+\frac{4\pi}{3})$; node 4 was the sum of nodes 1 through 3. The output of node 4 was sampled at 32 equal intervals between 0 and 2π.

The data from node 4 was preprocessed in the manner of Example D: the *ij*-th entry of the source matrix for node 4 was defined to be the absolute value of the difference between the *i*-th and *j*-th samples of the node 4 time series. A second, reference, source matrix was defined using the same preprocessing technique, now applied to thirty two equal interval samples of the function $\sin(x)$ for $0 \le x \le 2\pi$. The resulting 2, 32×32 source matrices were input to PROXSCAL with technical specification: sources- 2, objects- 32, dimension- 1 to 6, model- weighted, initial configuration- simplex, conditionality- conditional, transformations- numerical, rate of convergence- 0.0, number of iterations- 500, and minimum stress- 0.0. The dimension specification had a range of values, 1 to 6. The dimension resulting in the lowest stress/energy is the dimensionality of the underlying network.

Table 2, FIG. 10, shows dimension and corresponding stress/energy values from the analysis by the tool 100 of the 4-node network. The stress/energy minimum is achieved in dimension 4, hence the tool 100 has correctly determined network dimensionality. Similar experiments were run with more sophisticated dynamical systems and networks. Each of these experiments resulted in the successful determination of system degrees of freedom or dimensionality. These

experiments included the determination of the dimensionality of a linear feedback shift register. These devices generate pseudo-random bit streams and are designed to conceal their dimensionality.

From the foregoing, it can be seen that the illustrated embodiment of the present invention provides a method and apparatus for classifying input data. Input data are received and formed into one or more matrices. The matrices are processed using IDMDS to produce a stress/energy value, a rate or change of stress/energy value, a source space and a common space. An output or back end process uses analytical or visual methods to interpret the source space and the common space. The technique in accordance with the present invention therefore avoids limitations associated with statistical pattern recognition techniques, which are limited to detecting only the expected statistical pattern, and syntactical pattern recognition techniques, which cannot perceive beyond the expected structures. Further, the tool in accordance with the present invention is not limited to the fixed structure of neural pattern recognizers. The technique in accordance with the present invention locates patterns in data without interference from preconceptions of models or users about the data. The pattern recognition method in accordance with the present invention uses energy minimization to allow data to self-organize, causing structure to emerge. Furthermore, the technique in accordance with the present invention determines the dimension of dynamical systems from partial data streams measured on those systems through calculation of stress/energy or rate of change of stress/energy across dimensions.

While a particular embodiment of the present invention has been shown and described, modifications may be made. For example, PROXSCAL may be replaced by other IDMDS routines which are commercially available or are proprietary. It is therefore intended in the appended claims to cover all such changes and modifications which fall within the true spirit and scope of the invention.

ENERGY MINIMIZATION FOR CLASSIFICATION, PATTERN RECOGNITION, SENSOR FUSION, DATA COMPRESSION, NETWORK RECONSTRUCTION, AND SIGNAL PROCESSING

APPENDIX

SOURCE CODE FOR PRESENTLY PREFERRED EMBODIMENT OF THE INVENTION

STEP 1. PREPROCESSING SOURCE CODE

Mathematica code for step 1 of the invention: preprocessing.

Mathematica packages:

```
<< LinearAlgebra`MatrixManipulation`;
```

Packages for preprocessing:

```
(* Creates dissimilarity matrix from list of data vectors.  No missing values. *)

BeginPackage["MakeDissMat`"]
MakeDissMat::usage =
 "MakeDissMat[M] creates dissimilarity matrices from list of lists M."

Begin["`Private`"]
MakeDissMat[M_] := Module[{DissMat},
  DissMat[L_] := Module[{len = Length[L]},
    Table[Abs[L[[i]] - L[[j]]], {i, len}, {j, len}]
      ];
  If[MatrixQ[M],
   Print["Number of sources: ", Length[M]];
   Print["Number of objects: ", Length[M[[1]]]];
   Flatten[DissMat/@M, 1],
   Print["Number of sources: 1"];
   Print["Number of objects: ", Length[M]];
   DissMat[M]]
  ]
End[]
EndPackage[]
```

```
(* Creates dissimilarity matrices from ratings
  data with alternative distances and allowance for missing data *)

BeginPackage["MakeDissMissVal`"]
MakeDissMissVal::usage =
 "MakeDissMissVal[R,form,metric,mv,prnt] creates (no. of columns) source
   dissimilarity matrices from matrix R with possible missing values.  Set ms=1 to
   indicate missing values; ms=0 otherwise; metric specifies the distance function
   to be calculated.  R is assumed to have the form: objects-by-categories.
   Form specifications are: form=1, list of dissimilarity matrices;
   form=2, vector form, a single dissimilarity matrix; form=3, stacked
   dissimilarity matrices.  prnt=1 means print source and object count."

Begin["`Private`"]
MakeDissMissVal[R_, form_, metric_, mv_, prnt_] :=
 Module[{Rt, numobj, numsource, dissims, vectform, stackform, temp = 0},
  numobj = Length[R];
  If[Length[R[[1]]] == 0, Rt = Transpose[Map[List, R]], Rt = Transpose[R]];
  numsource = Length[Rt];
  If[prnt == 1,
  Print["Number of sources: ", numsource];
  Print["Number of objects: ", numobj]];
  dissims = Table[Table[If[(Rt[[k, i]] < 0 || Rt[[k, j]] < 0) && mv == 1 && i != j, -1, Which[
       metric == 1, If[(temp = Abs[Rt[[k, i]] - Rt[[k, j]]]) > 10^(-5), temp, 0], metric == 2,
       If[(temp = Log[Abs[Rt[[k, i]] - Rt[[k, j]]] + 1]) > 10^(-5), temp, 0], True, 0]],
     {i, numobj}, {j, numobj}], {k, numsource}];
  vectform = Table[Sqrt[(R[[i]] - R[[j]]) . (R[[i]] - R[[j]])], {i, numobj}, {j, numobj}];
  stackform = Join@@dissims;
  Which[form == 1, dissims // N, form == 2, vectform // N, form == 3, stackform // N]
  ];
End[]
EndPackage[]
```

```
(* Creates hybrid dissimilarity matrix *)

BeginPackage["Hybrid`"]
Needs["LinearAlgebra`MatrixManipulation`"];
Needs["MakeDissMissVal`"];
(* need to use this package since need -1 in border of matrix *)
Hybrid::usage =
 "Hybrid[L] creates hybrid dissimilarity matrices from list of data vectors L."

Begin["`Private`"]
Hybrid[L_] := Module[{output, toprows, restofrows},
  toprows = Map[{Join[{0}, #]}&, L];
  restofrows = MapThread[AppendRows[#1, #2]&,
    {Map[Transpose[{#}]&, L], MakeDissMissVal[Transpose[L], 1, 1, 1, 0]}];
  output = Flatten[MapThread[Join[#1, #2]&, {toprows, restofrows}], 1];
  Print["Number of sources: ", Length[L]];
  Print["Number of objects: ", Length[output[[1]]]];
  output
  ]
End[]
EndPackage[]
```

```
(* Creates Toeplitz proximity matrices *)

BeginPackage["MakeToeplitz`"]
  MakeToeplitz::usage =
 "MakeToeplitz[L] creates a Toeplitz proximity matrix from the list L."

Begin["`Private`"]
MakeToeplitz[M_] := Module[{Toeplitz},
Toeplitz[L_] := Module[{len = Length[L], size},
  size = 1 + len (len - 1) / 2;
 For[i = 1, i <= size, i++,
   For[j = 1, j <= size, j++,
    If[i == j, a[i, j] = 0.0, a[i, j] = L[[Abs[i - j]]]]
    ];
   ];
  Table[a[i, j], {i, size}, {j, size}]
  ];
  If[MatrixQ[M],
   Print["Number of sources: ", Length[M]];
   Print["Number of objects: ", Length[M[[1]]] + 1];
   Flatten[Toeplitz/@M, 1],
   Print["Number of sources: 1"];
   Print["Number of objects: ", Length[M] + 1];
   Toeplitz[M]]
  ]
End[]
EndPackage[]
```

```
(* Creates proximity matrices by populating symmetric matrix of appropriate size *)

BeginPackage["MakeSymMat`"]
  MakeSymMat::usage =
 "MakeSymMat[M] creates a set of symmetric matrices from the list of data vectors M
   through entrywise substitution into a symmetric matrix of appropriate size."

Begin["`Private`"]
MakeSymMat[M_] := Module[{},
 Print["Number of sources: ", Length[M]];
 Sym[V_] := Module[{outmat = {}, symlen = Length[V], k = 1, symsize},
  symsize = (Sqrt[8 symlen + 1] + 1) / 2;
  For[i = 1, i <= symsize, i++,
   For[j = 1, j < i, j++,
    a[i, j] = V[[k]]; a[j, i] = V[[k]]; k++;
    ];
   ];
  Table[If[i == j, 0, a[i, j]], {i, symsize}, {j, symsize}]
  ];
 Augment[L_] := Module[{auglen, n, temp},
  auglen = Length[L];
  n = auglen;
  temp = 1;
  While[2 auglen - n^2 - n <= 0, temp = n + 1; n--];
   Flatten[Join[L, Table[-1, {(temp (temp - 1) / 2) - auglen}]]]
  ];
  outmat = Flatten[Map[Sym[Augment[#]]&, M], 1];
  Print["Number of objects: ", Length[outmat[[1]]]];
  outmat
  ]
End[]
EndPackage[]
```

```
(* Creates distance matrices from sets of configurations of points *)

BeginPackage["Lp`"]
Lp::usage = "Lp[C,p] calulates Minkowski
   distance with exponent p on sets C of configurations of points."

Begin["`Private`"]
Lp[C_, p_] := Module[{metric},
metric[X_, a_] := Module[{len = Length[X]},
  (Partition[Plus@@Transpose[
         Flatten[Table[Abs[(X[[i]] - X[[j]])]^a, {i, len}, {j, len}], 1]], len])^
     (1/a) //
    N
  ];
  If[MatrixQ[C[[1]]],
   Print["Number of sources: ", Length[C]];
   Print["Number of objects: ", Length[C[[1]]]];
   Flatten[Map[metric[#, p]&, C], 1],
   Print["Number of sources: 1"];
   Print["Number of objects: ", Length[C]];
   metric[C, p]]
]
End[]
EndPackage[]


(* Output from preprocessing packages *)

Print["Test data: "];
test = {{1, 2, 3}, {1, 5, 9}};
test // TableForm
Print["MakeDissMat output on test data:"];
MakeDissMat[test] // TableForm
Print["MakeDissMissVal output on test data:"]
MakeDissMissVal[Transpose[test], 3, 1, 1, 1] // TableForm
Print["Hybrid output on test data:"]
Hybrid[test] // TableForm
Print["MakeToeplitz output on test data:"]
MakeToeplitz[test] // TableForm
Print["MakeSymMat output on test data:"]
MakeSymMat[test] // TableForm
Print["Lp output on test data:"]
Lp[test, 2] // TableForm

Test data:

1       2       3
1       5       9

MakeDissMat output on test data:

Number of sources: 2
```

```
Number of objects: 3

0       1       2
1       0       1
2       1       0
0       4       8
4       0       4
8       4       0

MakeDissMissVal output on test data:

Number of sources: 2

Number of objects: 3

0       1.      2.
1.      0       1.
2.      1.      0
0       4.      8.
4.      0       4.
8.      4.      0

Hybrid output on test data:

Number of sources: 2

Number of objects: 4

0       1       2       3
1       0       1.      2.
2       1.      0       1.
3       2.      1.      0
0       1       5       9
1       0       4.      8.
5       4.      0       4.
9       8.      4.      0

MakeToeplitz output on test data:

Number of sources: 2

Number of objects: 4

0.      1       2       3
1       0.      1       2
2       1       0.      1
3       2       1       0.
0.      1       5       9
1       0.      1       5
5       1       0.      1
9       5       1       0.

MakeSymMat output on test data:

Number of sources: 2

Number of objects: 3
```

```
0       1       2
1       0       3
2       3       0
0       1       5
1       0       9
5       9       0

Lp output on test data:

Number of sources: 1

Number of objects: 2
```

```
0           6.7082
6.7082      0
```

STEP 2. PROCESSING SOURCE CODE

Mathematica code for step 2 of the invention: processing.

Mathematica code for IDMDS via Alternating Least Squares and Singular Value Decomposition.

IDMDS via Alternating Least Squares (ALS)

Mathematica packages:

```
<< LinearAlgebra`Cholesky`;
<< Graphics`Graphics3D`;
<< Graphics`Graphics`;
<< Graphics`MultipleListPlot`;
```

Subpackages for IDMDS via ALS:

```
(* distance matrix package *)

BeginPackage["Dmatrix`"]
Dmatrix::usage =
 "Dmatrix[C] calculates the Euclidean interpoint distances of input configuration C."

Begin["`Private`"]
Dmatrix[C_] := Module[{numobj},
  numobj = Length[C];
  Table[Sqrt[(C[[i]] - C[[j]]) . (C[[i]] - C[[j]])], {i, numobj}, {j, numobj}] // N
  ]
End[]
EndPackage[]


(* Stress Package *)

BeginPackage["Stress`"]
Needs["Dmatrix`"];
Stress::usage = "Stress[P,W,X] calculates Kruskal's stress."

Begin["`Private`"]
Stress[P_, W_, X_] := Module[{},
 Plus@@Flatten[(Map[W * #&, P] - Map[Dmatrix, X]) ^2]
  ]
End[]
EndPackage[]
```

```
(* Guttman Package *)

BeginPackage["Guttman`"]
Needs["Dmatrix`"];
Guttman::usage =
 "Guttman[P,X,W] computes the update configuration via the Guttman transform."
Begin["`Private`"]
Guttman[P_, X_, W_] := Module[{B, D, d, dim},
D = Dmatrix[X];
dim = Length[X];
d = Length[P];
B = Table[
    If[i != j && D[[i, j]] != 0, -(W[[i, j]] * P[[i, j]]) / D[[i, j]], 0], {i, d}, {j, d}];
N[(1 / dim) * (B - Sum[DiagonalMatrix[B[[i]]], {i, d}]) . X]
]
End[]
EndPackage[]


(* Vmat Package *)

BeginPackage["VMat`"]
VMat::usage = "VMat[W] computes the p.s.d. V matrix from the weight matrix W."
Begin["`Private`"]
VMat[W_] := Module[{dim = Length[W], V},
V = Table[If[i != j, -W[[i, j]], 0], {i, dim}, {j, dim}];
V + Sum[DiagonalMatrix[(-1) * V[[i]]], {i, dim}]
]
End[]
EndPackage[]


(* UnitNorm Package *)

BeginPackage["UnitNorm`"]
UnitNorm::usage = "UnitNorm[A] takes the
   list of diagonals A and unit normalizes them so that (1/n)∑A*A=1."
Begin["`Private`"]
UnitNorm[A_] := Module[{},
  Map[Sqrt[Plus@@(A^2)]^(-1) * #&, A] * Sqrt[Length[A]] // N
  ]
End[]
EndPackage[]
```

```
(* TMat Package *)

BeginPackage["TMat`"]
TMat::usage = "TMat[A] defines the T matrix which normalizes common space Z."
Begin["`Private`"]
TMat[A_] := Module[{},
  DiagonalMatrix[Sqrt[Plus@@(A^2)] * ((Sqrt[Length[A]])^(-1))] // N
  ]
End[]
EndPackage[]


(* InitialConfig Package *)

BeginPackage["InitialConfig`"]
InitialConfig::usage = "InitialConfig[ns, no, d] creates ns=number of sources, no=
   number of objects by d-dimensional constrained random start configurations."
Begin["`Private`"]
InitialConfig[ns_, no_, d_] :=
 Module[{numsources = ns, numobs = no, dimens = d, i, j, k, G, W},
  G = N[Table[(*SeedRandom[i*j];*)Random[], {i, numobs}, {j, dimens}]];
  W = N[Table[(*SeedRandom[k*j];*)Random[], {k, numsources}, {j, dimens}]];
  {Table[G.DiagonalMatrix[W[[k]]], {k, numsources}], G, W}
]
End[]
EndPackage[]


(* Diagonal Package *)

BeginPackage["Diagonal`"]
Diagonal::usage = "Diagonal[M] creates a diagonal matrix from main diagonal of M."
Begin["`Private`"]
Diagonal[M_] := Module[{dim},
  dim = Length[M];
  Table[If[i == j, M[[i, j]], 0], {i, dim}, {j, dim}] // N
]
End[]
EndPackage[]
```

```
(* UnDiagonal Package *)

BeginPackage["UnDiagonal`"]
UnDiagonal::usage = "UnDiagonal[M] turns diagonal matrix into a vector."
Begin["`Private`"]
UnDiagonal[M_] := Module[{dim},
  dim = Length[M];
  Table[M[[i, i]], {i, dim}] // N
]
End[]
EndPackage[]
```

IDMDS via ALS:

```
(* IDMDS via ALS *)

BeginPackage["IDMDSALS`"]
Needs["Stress`"];
Needs["InitialConfig`"];
Needs["UnitNorm`"];
Needs["VMat`"];
Needs["TMat`"];
Needs["Guttman`"];
Needs["Diagonal`"];
Needs["UnDiagonal`"];
IDMDSALS::usage = "IDMDSALS[prox_,proxwts_,dim_,epsilon_,
   iterations_,seed_] computes IDMDS for proximity matrices prox."
Begin["`Private`"]
IDMDSALS[prox_, proxwts_, dim_, epsilon_, iterations_, seed_] := Module[
  {Aconstrain, A0, BX, deltastress, deltastresslist, numits, numobj, numscs, stresslist,
   T, T0, tempstress, tempstressprev, V, Vin, Xupdate, X0, Zconstrain, Zt, Z0},
  numobj = Length[prox[[1]]];
  numscs = Length[prox];
  numits = 0;
  SeedRandom[seed];
  {X0, Z0, A0} = InitialConfig[numscs, numobj, dim];
   Print["Number of sources:  ", numscs];
   Print["Number of objects:  ", numobj];
  T0 = TMat[A0];
  Z0norm = Z0 . T0;
  A0norm = Map[Inverse[T0] . #&, Map[DiagonalMatrix, A0]];
  X0 = Map[Z0norm . #&, A0norm];
  V = VMat[proxwts];
  Vmp = PseudoInverse[V];
  tempstress = Stress[prox, proxwts, X0];
  stresslist = {tempstress};
  deltastress = tempstress;
  deltastresslist = {};
```

```
While[deltastress > epsilon && numits <= iterations,
 numits++;
 Xupdate = MapThread[Guttman[#1, #2, proxwts]&, {prox, X0}];

 For[i = 1, i <= 1, i++,
 Zconstrain = (1 / numscs) * (Vmp . Plus@@MapThread[#1 . #2&, {numobj * Xupdate, A0norm}]);
 Zt = Transpose[Zconstrain];
 Aconstrain = Map[Inverse[Diagonal[Zt . V . Zconstrain]] . #&,
    Map[Diagonal, Map[Zt . #&, numobj * Xupdate]]];

 T = TMat[Map[UnDiagonal, Aconstrain]];

 A0norm = Map[Inverse[T] . #&, Aconstrain];
 Z0norm = Zconstrain . T;
  ];

 X0 = Map[Z0norm . #&, A0norm];

 tempstressprev = tempstress;
 tempstress = Stress[prox, proxwts, X0];
 stresslist = Append[stresslist, tempstress];
 deltastress = tempstressprev - tempstress;
 deltastresslist = Append[deltastresslist, deltastress];
 ];
Print["Final stress is:  ", tempstress];
Print["Stress record is:  ", stresslist];
Print["Stress differences:  ", deltastresslist];
Print["Number of iterations:  ", numits];
Print["The common space coordinates are:  ", Z0norm // MatrixForm];
Print["The source space coordinates are:  ", Map[MatrixForm, Chop[Aconstrain]]];
  {Z0norm, Chop[Aconstrain]}
  ]
End[]
EndPackage[]
```

IDMDS via Singular Value Decomposition (SVD)

Mathematica packages:

```
<< Graphics`Graphics3D`;
<< Graphics`Graphics`;
```

Subpackages for IDMDS via SVD:

```
(* DistanceMatrix Package *)

BeginPackage["DistanceMatrix`"]
DistanceMatrix::usage = "DistanceMatrix[
   config] produces a distance matrix from configuration matrix config."
Begin["`Private`"]
DistanceMatrix[config_] := Module[{c, d, m, one, temp},
  d = Length[config];
  m = config . Transpose[config];
   c = {Table[m[[i, i]], {i, d}]};
   one = {Table[1, {i, d}]};
N[Sqrt[ Transpose[one] . c + Transpose[c] . one - 2 m]]
 ]
End[]
EndPackage[]



(* DiagMatNorm Package *)

BeginPackage["DiagMatNorm`"]
DiagMatNorm::usage = "DiagMatNorm[A] normalizes the list of weight vectors A."
Begin["`Private`"]
DiagMatNorm[A_List] := Module[{newA, norm, sumnorm = 0},
  newA = Table[0, {Length[A]}, {Length[A[[1]]]}];
   For[i = 1, i <= Length[A[[1]]], i++,
    For[j = 1, j <= Length[A], j++,
    sumnorm += A[[j, i]]^2;
    ];
   norm[i] = Sqrt[sumnorm];
   sumnorm = 0;
   ];
  norm = Table[norm[k], {k, Length[A[[1]]]}];
  For[i = 1, i <= Length[A[[1]]], i++,
     For[j = 1, j <= Length[A], j++,
    newA[[j, i]] = A[[j, i]] / norm[[i]];
    ];
   ];
  N[newA]
 ]
End[]
EndPackage[]
```

```
(* NormalizeG *)

BeginPackage["NormalizeG`"]
NormalizeG::usage = "NormalizeG[A] gives matrix
   which normalizes the common space given the list of weight vectors A."
Begin["`Private`"]
NormalizeG[A_List] := Module[{newA, norm, summorm = 0},
  newA = Table[0, {Length[A]}, {Length[A[[1]]]}];
   For[i = 1, i <= Length[A[[1]]], i++,
    For[j = 1, j <= Length[A], j++,
    summorm += A[[j, i]]^2;
    ];
   norm[i] = Sqrt[summorm];
   summorm = 0;
   ];
  norm = Table[norm[k], {k, Length[A[[1]]]}];
  N[DiagonalMatrix[norm]]
]
End[]
EndPackage[]


(* BMatrix Package *)

BeginPackage["BMatrix`"]
BMatrix::usage = "BMatrix[delta, DM] is part of the Guttman transform."
Begin["`Private`"]
BMatrix[delta_, DM_] := Module[{b, bdiag, d, i, j, k},
  d = Length[delta];
  b = Table[0, {d}, {d}] // N;
    For[i = 1, i <= d, i++,
      For[j = 1, j <= d, j++,
        If[i != j && DM[[i, j]] != 0, b[[i, j]] = b[[i, j]] - delta[[i, j]] / DM[[i, j]]];
  ];
 ];
bdiag = Sum[DiagonalMatrix[b[[k]]], {k, d}];
N[b - bdiag]
]
End[]
EndPackage[]
```

```
(* GuttmanTransform Package *)

BeginPackage["GuttmanTransform`"]
GuttmanTransform::usage = "The GuttmanTransform[B, X]
   updates the configuration X through multiplication by the BMatrix B."
Begin["`Private`"]
GuttmanTransform[B_, X_] := Module[{dim},
  dim = Length[X];
 N[ 1/dim B . X]
]
End[]
EndPackage[]


(* AGWStress Package *)

BeginPackage["AGWStress`"]
AGWStress::usage = "The AGWStress[dissimilarity,
   distance] is the loss function for multidimensional scaling."
Begin["`Private`"]
AGWStress[dissimilarity_, distance_] := Module[{S, dim, i , j},
   dim = Length[dissimilarity];
  S = 0;
       For[j = 1, j <= dim, j++,
        For[i = 1, i < j, i++,
            S += (dissimilarity[[i, j]] - distance[[i, j]]) ^2
           ];
         ] ;
 N[S]
  ]
End[]
EndPackage[]
```

```
(* NormStress *)

BeginPackage["NormStress`"]
NormStress::usage = "NormStress[dissimilarity] normalizes the stress loss function."
Begin["`Private`"]
NormStress[dissimilarity_List] := Module[{norm, numobjects, i , j},
  numobjects = Length[dissimilarity[[1]]];
  norm = 0.0;
  For[j = 1, j <= numobjects, j++,
       For[i = 1, i < j, i++,
           norm += dissimilarity[[i, j]]^2
          ];
        ];
 N[norm]
  ]
End[]
EndPackage[]


(* InitialConfig2 Package *)

BeginPackage["InitialConfig2`"]
Initialconfig::usage = "Initialconfig[ns, no, d] creates ns=number of sources, no=
   number of objects by d-dimensional constrained random start configurations."
Begin["`Private`"]
Initialconfig[ns_, no_, d_] :=
 Module[{numsources = ns, numobs = no, dimens = d, i, j, k, G, W},
  G = N[Table[(*SeedRandom[i*j];*)Random[], {i, numobs}, {j, dimens}]];
  W = N[Table[(*SeedRandom[k*j];*)Random[], {k, numsources}, {j, dimens}]];
  Table[G.DiagonalMatrix[W[[k]]], {k, numsources}]
]
End[]
EndPackage[]
```

```
(* Torgerson Package *)

BeginPackage["Torgerson`"]
Torgerson::usage = "Torgerson[D,d] computes classical
   2(d) dimensional scaling solution for dissimilarity matrix D."
Begin["`Private`"]
Torgerson[D_, d_] := Module[{Dsq, u, v, w, Bdelta, J, One, n = Length[D]},
  One = {Table[1, {i, n}]};
  J = IdentityMatrix[n] - (1 / n) Transpose[One] . One;
  Dsq = N[Map[#^2 &, D, 1]];
  Bdelta = N[(-0.5) J . Dsq . J];
  {u, w, v} = SingularValues[Bdelta, Tolerance -> 0];
  Transpose[Table[v[[i]], {i, d}]] . DiagonalMatrix[Table[w[[i]], {i, d}]] // Chop
  ]
End[]
EndPackage[]


(* Ave Package *)

BeginPackage["Ave`"]
Ave::usage = "Ave[M] finds the average of the list of matrices M and
   produces a list of the same length with every element the average of M."
Begin["`Private`"]
Ave[M_List] := Module[{},
average = N[(1 / Length[M]) * Apply[Plus, M]];
Table[average, {i, Length[M]}]
  ]
End[]
EndPackage[]
```

```
(* SVDConstrain Package *)

BeginPackage["SVDConstrain`"]
SVDConstrain::usage =
 "SVDConstrain[configs] imposes constraints on the list of configurations configs."
Begin["`Private`"]
SVDConstrain[configs_List, d_] := Module[{numsrcs = Length[configs],
   numobjects = Length[configs[[1]]], dim = d, y, u, v, uhold, vhold, G, weights},
For[i = 1, i <= dim, i++,
    y[i] = Transpose[Table[Map[Transpose, configs][[k, i]], {k, numsrcs}]];
   ]; Print[Table[y[i], {i, dim}]];

For[i = 1, i <= dim, i++,
   {uhold[i], w[i], vhold[i]} = N[SingularValues[N[y[i]], Tolerance -> 0] // Chop];
   ];
  u = Table[uhold[i], {i, dim}];
  v = Table[vhold[i], {i, dim}];
  G = N[Transpose[Table[u[[i, 1]], {i, dim}]]];
  weights =
   N[Partition[Flatten[Table[w[k][[1]] * v[[k, 1, j]], {j, numsrcs}, {k, dim}]], dim]];
  Table[G . Map[DiagonalMatrix, weights][[i]], {i, numsrcs}]
 ]
 End[]
 EndPackage[]
```

IDMDS via SVD:

```
(* IDMDS via SVD on constraints with implicit normalization of stress. *)

BeginPackage["IdmdsSVD`"]
Needs["DistanceMatrix`"];
Needs["BMatrix`"];
Needs["GuttmanTransform`"];
Needs["AGWStress`"];
Needs["NormStress`"];
Needs["DiagMatNorm`"];
Needs["NormalizeG`"];
Needs["InitialConfig2`"];
Needs["Torgerson`"];
Needs["SVDConstrain`"];
Needs["Ave`"];
Needs["Graphics`Graphics`"];
Needs["Graphics`Graphics3D`"];
IdmdsSVD::usage =
  "IdmdsSVD[
   diss, dim, iterations, rate, start] is an IDMDS package for an arbitrary  number
   of sources.  Enter a list of dissimilarity matrices = diss; the dimension =
   dim of the common space; the number of iterations required = iterations;
   the rate of convergence = rate.  Has random, start=1;  Torgerson,
```

```
   start=2; or user-provided, start=startlist start configurations."
Begin["`Private`"]
IdmdsSVD[diss_,  dim_,  iterations_ , rate_, start_] :=
 Module[{ d = dim, ϵ = rate, G, Gnorm, gt, holdstress, iterate,
   konstant, numit = iterations, numobjects,  numsrcs = Length[diss], sdr,
  sr, stress, stressdiffrecord, stressrecord, temp, tempstress,
   u, updates, utemp, v, vtemp, w, weightrecord, weights, weightsnorm, wr},
   numobjects = Length[diss[[1]]];
   tempstress = 0.0;
   holdstress = 0.0;
   stress = 0.0;
   konstant = 0.0;
   iterate = 0;

  For[i = 1, i <= numsrcs, i++,
            konstant += NormStress[diss[[i]]];
     ];
 If[NumberQ[start] && start == 1, temp = InitialConfig2[numsrcs, numobjects, d]];
 If[NumberQ[start] && start == 2, temp = SVDConstrain[Map[Torgerson[#, d]&, diss], d]];
 If[NumberQ[start] && start == 3, temp = SVDConstrain[Map[Torgerson[#, d]&, Ave[diss]], d]];
 If[NumberQ[start] == False,  temp = start];

     For[i = 1, i <= numsrcs, i++,
         holdstress += AGWStress[diss[[i]], DistanceMatrix[temp[[i]]]
     ];
    ];
      If[konstant != 0,
         stress = holdstress / konstant,
         stress = holdstress];
      Print["Initial stress is:  ", stress];

   sdr = {0.0};
   sr = {stress};
   wr = {};

   While[ iterate <= numit,

    For[i = 1, i <= numsrcs, i++,
        gt[i] =
      GuttmanTransform[BMatrix[diss[[i]], DistanceMatrix[temp[[i]]]], temp[[i]]];
       ];

    X = Table[gt[i], {i, numsrcs}];

       For[i = 1, i <= d, i++,
          y[i] = Transpose[Table[Map[Transpose, X][[k, i]], {k, numsrcs}]];
        ];

     For[i = 1, i <= d, i++,
     {utemp[i], w[i], vtemp[i]} = N[SingularValues[y[i]]];
     ];
```

```
  u = Table[utemp[i], {i, d}];
  v = Table[vtemp[i], {i, d}];

 G = N[Transpose[Table[u[[i, 1]], {i, d}]]];
        (*Print["G is:  ",MatrixForm[G]];*)

weights = N[Partition[Flatten[Table[w[k][[1]] * v[[k, 1, j]], {j, numsrcs}, {k, d}]], d]];

temp = Table[G . Map[DiagonalMatrix, weights][[i]], {i, numsrcs}];

   tempstress = 0.0;

    For[i = 1; holdstress = 0.0, i <= numsrcs, i++,
         holdstress += AGWStress[diss[[i]], DistanceMatrix[temp[[i]]]];
        ];
     If[konstant != 0,
        tempstress = holdstress / konstant,
        tempstress = holdstress];

   weightrecord = AppendTo[wr, weights];
   stressrecord = AppendTo[sr, tempstress];
   stressdiffrecord = AppendTo[sdr, stress - tempstress ];

   If[stress - tempstress <= ϵ, Break[],
    stress = tempstress;
    iterate++;
    ];  (* end If *)
   ];    (* end while *)
  Gnorm = G . NormalizeG[weights];
  weightsnorm = DiagMatNorm[weights];
  For[j = 1, j <= numsrcs, j++,
    For[i = 1, i <= d, i++,
     If[weightsnorm[[j, i]] < 0,
      weightsnorm = ReplacePart[weightsnorm, -weightsnorm[[j, i]], {j, i}];
      ];
     ];
    ];

Print["Number of iterations was:  ", iterate];
Print["Final stress:  ", tempstress];
Print["Stress difference record:  ", stressdiffrecord];
Print["Stress record:  ", stressrecord];
Print["Coordinates of the common space:"];
Print[MatrixForm[Gnorm]];
Print["Coordinates of the weight space:  "];
Print[MatrixForm[weightsnorm]];
(*Print["Coordinates of the private spaces:  "];
Print[Map[MatrixForm,Map[G.#&,Map[DiagonalMatrix,weightsnorm]]]];*)
Print["Plot of common space:"];
  Which[d == 3, ScatterPlot3D[Gnorm, PlotStyle -> {{AbsolutePointSize[5]}}],
```

```
    d == 2, TextListPlot[Gnorm], d == 1,
    ListPlot[Flatten[Gnorm], PlotJoined -> True, Axes -> False, Frame -> True];
    ];
Print["Plot of the source space:"];
  Which[d == 3, ScatterPlot3D[weightsnorm, PlotStyle -> {{AbsolutePointSize[5]}},
    ViewPoint -> {2.857, 0.884, 1.600}(*,ViewPoint->{1.377, 1.402, 2.827}*)], d == 2,
    TextListPlot[weightsnorm, PlotRange -> All, Axes -> False, Frame -> True], d == 1,
    ListPlot[Flatten[weightsnorm], PlotJoined -> True, Axes -> False, Frame -> True];
    ];
]
End[]
EndPackage[]
```

STEP 3. POSTPROCESSING SOURCE CODE

Mathematica code for step 3 of the invention: postprocessing.

Subfunctions:

```
(* Euclidean distance between vectors u and v *)

d[u_, v_] := Sqrt[(u - v) . (u - v)] // N;


(* Characteristic function *)

char[u_, x_] := If[x === u, 1, 0];
```

Clustering functions for back-end analysis of classifying space of invention:

```
g1[obj_, dyeval_, dyelist_, L1_, L2_, power_] := Plus@@Table[If[i == obj, 0,
     char[dyeval, dyelist[[i]]] d[L1[[obj]], L2[[i]]] ^ (power)], {i, Length[dyelist]}] /
                    Plus@@
   Table[If[i == obj, 0, d[L1[[obj]], L2[[i]]] ^ (power)], {i, Length[L2]}];

g2[obj_, dyeval_, dyelist_, L2_, power_] := Plus@@Table[If[i == obj, 0,
     char[dyeval, dyelist[[i]]] d[L2[[obj]], L2[[i]]] ^ (power)], {i, Length[dyelist]}] /
                    Plus@@
   Table[If[i == obj, 0, d[L2[[obj]], L2[[i]]] ^ (power)], {i, Length[L2]}];

g3[obj_, dyeval_, dyelist_, L2_, power_] :=
 Plus@@Table[If[i == obj, 0, char[dyeval, dyelist[[i]]]
      Exp[d[L2[[obj]], L2[[i]]]] ^ (power)], {i, Length[dyelist]}] /
                    Plus@@
   Table[If[i == obj, 0, Exp[d[L2[[obj]], L2[[i]]]] ^ (power)], {i, Length[L2]}];
```